Aalborg Universitet

AALBORG UNIVERSITY
DENMARK

New Tools For Research and Development Acceleration of GNSS Receivers

Plausinaitis, Darius; Borre, Kai

Published in:
Proceedings of 5th ESA Workshop on Satellite Navigation Technologies and European Workshop on GNSS Signals and Signal Processing (NAVITEC)

DOI (link to publication from Publisher):
10.1109/NAVITEC.2010.5707992

Publication date:
2010

Document Version
Accepted author manuscript, peer reviewed version

Link to publication from Aalborg University

Citation for published version (APA):
Plausinaitis, D., & Borre, K. (2010). New Tools For Research and Development Acceleration of GNSS Receivers. In *Proceedings of 5th ESA Workshop on Satellite Navigation Technologies and European Workshop on GNSS Signals and Signal Processing (NAVITEC)* IEEE Press.
https://doi.org/10.1109/NAVITEC.2010.5707992

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal -

Take down policy
If you believe that this document breaches copyright please contact us at vbn@aub.aau.dk providing details, and we will remove access to the work immediately and investigate your claim.

Downloaded from vbn.aau.dk on: May 05, 2023

New Tools For Research and Development Acceleration of GNSS Receivers

Darius Plausinaitis
Danish GPS Center
Aalborg University, Denmark
Email: dpl@gps.aau.dk

Kai Borre
Danish GPS Center
Aalborg University, Denmark
Email: borre@gps.aau.dk

***Abstract*—This paper analyzes strengths and shortcomings of the existing aids for GNSS receiver development. A new set of tools is presented that provides to the developer debugging and optimization capabilities not supplied so far by existing tools. The proposed tools are simple and therefore very affordable, but never the less they are very functional.**

The first results are presented from the application of the proposed tools in a GNSS receiver debugging.

I. Introduction

Real-time GNSS receiver development in an embedded-type platform has its own development challenges. Most general purpose software debugging tools are oriented into traditional software application debugging, but often are not convenient to debug signal processing aspects of the receiver. Furthermore due to receiver requirements, usually the CPUs are slow and memory for data or code is very limited. This limits the choice of debugging aids that can be implemented as receiver code or possibilities for in-receiver debugging data storage, event logging. All this limits how long or what data or signals can be observed in the receiver under development or testing.

One more problem for receiver development is the availability of required GNSS test signals. Dedicated hardware based GNSS signal simulators and test beds are most often used to perform complete receiver tests. Both of these are expensive solutions. Not all companies or institutions can afford to acquire these necessary tools or access to the tools is time limited and creates other inconveniences. The next alternative—the signals in space might not be available all the time. For example today GIOVE signals are only visible to the user once, twice a day. Out of these signal visibility cases some signals will be obstructed due to low elevation angles.

We start with an outline of the available GNSS receiver debugging options and their drawbacks. Then we present the basic debugging technique proposed by DGC. In the later sections we add more features to this initial, basic setup. In the last part of the paper we outline other applications of the proposed GNSS debugging tools.

II. Current receiver debugging facilities

There a number of existing software and hardware based tools when it comes to debugging and optimization of GNSS receivers or embedded systems in general.

During development the engineers will build the software and hardware in incremental manner from smaller blocks and then use test vectors to verify correct functionality of each separate block. Because each building block can be isolated from effects of other blocks and tested with known data the testing here is relatively easy and does not create especially big challenges. Challenges arise when the complete system must be tested. This is caused by interrelations between all building blocks, execution in real-time and possibly when receiving live GNSS signals in difficult reception conditions. This was encountered also during the DGC development of GNSS receivers.

In the rest of this section we outline which information from the receiver would be helpful and the possible aids in receiver testing and debugging and the drawbacks of the already available choices. Only the tools that are the most popular and the most relevant to the topic of the proposed solution are presented.

A. A short anatomy of the patient

First we would like to outline which information is helpful to the engineer when he is debugging a GNSS receiver. An engineer (or a group of engineers from different fields) first of all needs to validate correct functionality of the receiver. This requires some test input signals and some test measurements in various places along the signal processing chain. To do this an access is needed to some key points of the GNSS receiver as is shown in Figure 1.

In the figure we see typical receiver internal details and the key points in the receiver signal processing chain where receiver operation can be monitored. The description of receiver operation can be found for example in [2] and will be omitted in this paper.

Point 1 can be connected to a GNSS signal simulator instead of antenna to provide tests signals for the receiver.

At point 2 the correct functionality of the receiver front-end can be validated.

Signals from points 3 and 4 would be very useful as they allow to see the actual correlation results and also which values are fed back by the loop close software. From the correlation values it is possible to observe the actual tracking noise, steady state errors or any anomalies in the received signals. Interference effects and channel software responses to it also

978-1-4244-8739-4/10/$26.00 ©2010 IEEE

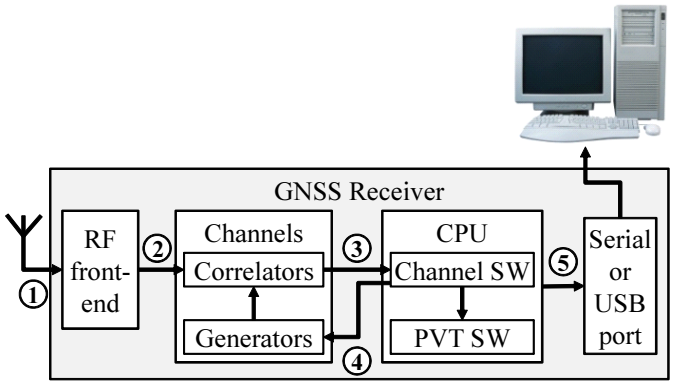

Fig. 1. The key measurement points along the GNSS receiver signal processing chain.

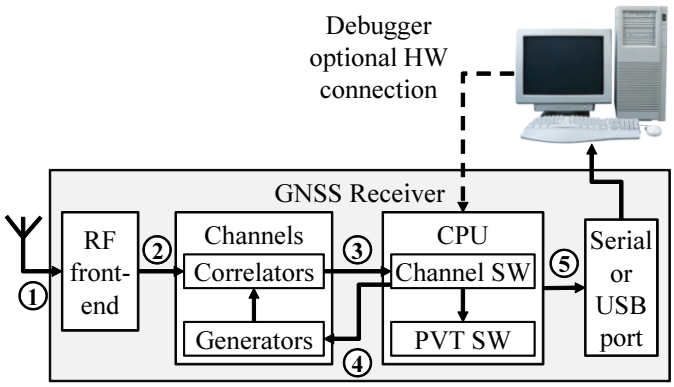

Fig. 2. GNSS receiver test setup with a debugger.

could be monitored. Unfortunately these very useful signals (and crucial in channel development) are the most difficult to monitor. One of the reasons is that these signals are not routed outside the receiver ASIC. The second reason is that the physical signal transfer is implemented in a bus and is mixed with data from other receiver devices (for example serial port, timers etc.) that travel to and from the receiver's CPU.

A bus is a group of signals, wires used to interconnect two or more devices. It enables data exchange between these devices. The bus itself is composed of two main subgroups of signals. One is used to transfer actual data and the second is used to indicate the destination or the source address of the transfered data. Commonly these two subgroups are called data bus and address bus.

In a simple bus there can be only one active data transfer at any instance of time. Devices "take turns" to send data through the bus that is shared between them. This means that the stream of transfer operations in the bus will contain a mix of data (in time domain) coming from different sources.

In this paper a single event on the bus, like a data transfer to the processor or a transfer from the processor, will be called a bus transaction.

Receiver PVT solution messages and other status data are sent by the receiver software through point 5. The receiver developer can modify the receiver software to supply virtually any data that is available to the CPU. But in practice the amount or rate of such data can be severely limited by the connection speed to the PC or the receiver's CPU performance.

B. Debuggers

A debugger is a software tool that allows to stop and inspect a program running on a CPU. The debugger itself can run on the same CPU or externally, for example on a PC. In the latter case there must be some hardware means that enables this PC to communicate with the CPU under test. The debugger can use a dedicated hardware connection to the receiver, or it can use the existing serial, or USB data connection for the same purpose. The debugger is the primary way to test any system that contains a CPU. The setup with a debugger is shown in Figure 2.

Apart from of code inspection, the debugger allows to read data from the receiver memory. This means that any data that are used by the CPU can be also accessed by the debugger. This corresponds to access data at points 3, 4, and 5 as shown in the figure. But this can only be done when the receiver software is temporally stopped. Unfortunately for a GNSS receiver this stop brakes the signal processing as the channels are not stopped, and the incoming signal is arriving continuously. After such stop the receiver will have to restart its normal operation from the acquisition step.

It is possible to log the data of interest (for example the correlation values) in the receiver memory. Additional code must be inserted into receiver software to do this data logging. After the logging is complete the receiver program is stopped and the debugger then can read the complete log of the data. Or alternatively the receiver CPU could send this data through the existing connection to the PC (point 5 in the figures). But in this case the receiver CPU must have processing power surplus to handle this relatively high speed data transfer.

Such data storage solution requires substantial amounts of receiver memory, especially if data from all channels must be logged for minutes. For a single GPS channel which is based on six 32 bits wide correlators the created data rate is about 24 KB/s or 1.44 MB/min. This number has to be multiplied by the number of monitored channels. This is especially a clearly visible problem for receivers that use on chip memory with size a few hundreds of kilobytes (such type of memory is expensive and therefore as small as possible). This allows to capture short snapshots of receiver operation. Furthermore the receiver operation is still interrupted by the debugger, although not constantly.

An additional problem is that due to limited memory space some conditions must be placed in the receiver software when to activate the logging of data. It is not always possible in the receiver software to detect the conditions or the events that shall activate data logging (for example a too large deviation from the true trajectory or a particular behavior of a channel). If the cause of the receiver problems is not known, it becomes even more tedious to define conditions when to log the receiver data.

In addition to these inconveniences the additional code and storage of log data can alter the receiver behavior. This is because the additional code alters the length of software procedures (in particularly this is critical for code that is serving channels) and also because some software bugs are sensitive to the memory layout of the receiver software. In such cases the previously observed receiver problems can disappear or change their manifestation patterns.

C. Logic analyzers

Logic analyzers are dedicated tools to visualize and to log digital signals. For a not familiar reader it can be described as an equivalent of an electrocardiography device for monitoring of digital circuits. It can show the actual binary values of digital signals as well as their changes in a time plot. The logic analyzers are always used to debug and verify the GNSS receiver hardware (and also any other kind of digital devices).

The advanced (and expensive) logic analyzers can show and log more than 100 signals and can contain capabilities to directly decode the bus activity of a CPU. Also it can have DSP related capabilities for example to do FFT and other kinds of signal plots (alone or if connected to additional equipment). This feature allows the analyzer to plot signal spectrum of the receiver IF signal from test point 2 shown in Figure 1.

In an FPGA based receiver development it is possible to monitor with logic analyzer virtually any signal inside of the FPGA chip. If ASICs are used, then only signals at the outputs of the chips can be monitored, unless the ASIC design includes special facilities to access internal signals.

Although the logic analyzers can interpret the actual values of the data words sent between CPU and the channels (points 3 and 4 in Figure 1), it will have trouble to sort out which data at what time instance belongs to which channel and even to what particular component of this channel. This means that it cannot present the captured data at points 3 and 4 in a unique way.

D. GNSS signal simulators

A GNSS simulator (or a generator) is used to generate a precisely specified set of GNSS signals as they would be observed by a receiver moving along a predefined track. The signals at the GNSS simulator output are at RF frequency (L1, E5 etc.). Typically the simulation includes atmosphere effects, multipath, and to some degree other effects. The output of a GNSS simulator is connected to the receiver instead of the receiver GNSS antenna (at point 1 in Figure 3). Then the receiver computed position, velocity, and time data from point 5 is compared to reference data supplied by the simulator.

Hardware GNSS simulators use precisely calibrated software and hardware to simulate in real-time as precise as possible signals at RF. Therefore good GNSS simulators are expensive.

There is a different class of GNSS simulators that do not use hardware to generate signals. These are pure software based simulators. Some of them simulate complete GNSS systems from end to end. This means that the signals are simulated

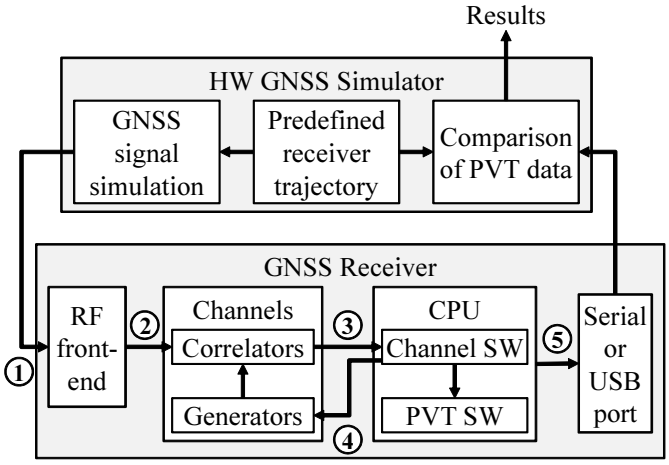

Fig. 3. GNSS receiver test setup with a simulator.

all the way from satellite constellation and propagation to the computation of the receiver position. Any intermediate signals in this simulation chain can be generated including the sampled IF signals after receiver front-end.

Such simulators cannot be used for a real, non PC based GNSS receiver tests as there is no direct way to interface the simulated signals to the actual receiver hardware. These simulators are mostly used for research and development purposes.

Hardware simulators are very good to verify GNSS receiver operation and to detect or to recreate the problematic cases. But on their own, they do not provide any additional insight into the receiver signal processing part and cannot aid directly in pinpointing to the malfunctioning part of the receiver.

E. Missing information

Out of the listed existing development and debugging aids none can provide a convenient means to access the intermediate results of GNSS signal processing.

Signals at point 2 can be accessed even in some receiver ASICs, but this is usually only necessary during optimization of the developed receiver front-end.

Signals from points 3 and 4 are the most valuable in channel debugging and the most difficult to access.

All data in the receiver CPU and the receiver memory can be accessed by a software debugger tool, but the debugger access can interrupt signal processing.

In view of these problems the DGC has developed new aiding tools to access information at points 3 and 4 as shown in the figures in this section. The proposed solution is transparent to the receiver signal processing chain and therefore does not interrupt it.

III. The basic DGC debugging solution

The basic DGC solution is a bit similar to the logic analyzer approach, but the DGC solution is using a dedicated design which makes the hardware very simple and cheap. The more complex data processing and analysis software are executed

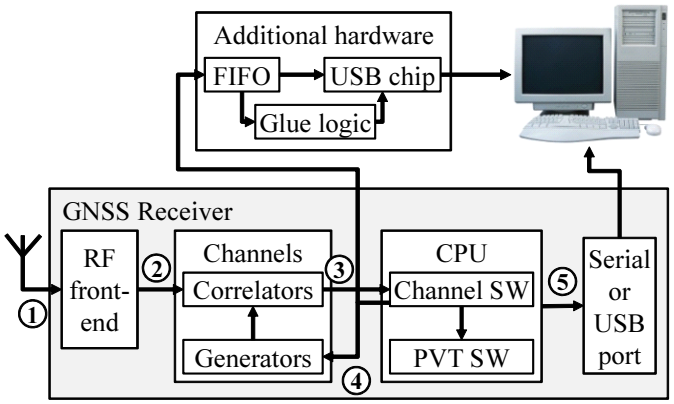

Fig. 4. Basic version of the proposed GNSS receiver debugging setup.

on a PC. The signals are captured only at the instance of bus transaction to or from the channels. Figure 4 shows the DGC debugging setup.

The points 3 and 4 in the figure physically are implemented in a form of a data bus and both data transfers travel through this single bus. This means that only a copy of bus signals (data and part of address lines and a transfer direction flag) must be made to get access to both the correlation information and the loop closure data. This bus transaction copy is stored in a FIFO buffer. From there this information is fetched by a dedicated USB chip and sent over a USB connection. The USB data stream is captured by a software running on a PC. The captured data then can be stored in a file for post processing or can be analyzed and displayed in real-time.

The software analyzes the captured data by inspecting one receiver CPU transfer at a time. First it must decode from the captured address lines the channel and the component of the channel that is accessed by this data transfer. To do this, the configuration of the receiver address space must be known to the software. Then the actual captured data can be appended to an array that is associated to this component (NCO increments, correlator values). At the end the time plots are made of each from these arrays or data analysis is performed. For example the variance of the code tracking noise can be computed from increment values that is sent to the code NCO.

In practice it means also that any other data exchange on the same bus between CPU and any other peripherals (like serial ports, timers) can be captured and presented by the proposed tools if needed.

IV. The complete debugging solution

In the augmentation of the basic GNSS debugging setup we address the issue of test GNSS signals. The most reliable testing conditions are obtained by use of GNSS simulators. There are two problems associated with signal simulators.

One is the high price. Even if the simulator is acquired, it can happen that it has to be sheared across members of the development team as it is too expensive to acquire a GNSS simulator per developer.

TABLE I
An example of visibility predictions for GIOVE satellites at the DGC site

Date	Time	Satellite	Maximum elevation [°]
28/10/2010	12:51:30	GIOVE-B	71.6
28/10/2010	18:26:30	GIOVE-A	71.9
29/10/2010	04:04:07	GIOVE-B	11.6
29/10/2010	06:39:42	GIOVE-A	35.2
29/10/2010	18:23:45	GIOVE-B	85.5
29/10/2010	22:42:20	GIOVE-A	30.6

The second problem is that although simulators can simulate many real life signal propagation conditions, they are unable to simulated all cases. Besides this, in real life various kinds of interferences can be associated with each particular receiver site. Therefore the receiver can exhibit unstable operation, although it has successfully passed tests with a simulator.

An alternative solution is to use live GNSS signals. This solution has its own problems. First, there is no control and no or little knowledge about true parameters of the received signals. Also it can be hard to recreate the signal reception conditions for repeated tests.

The second problem is the availability of signals. For example GIOVE or COMPASS signals may be available just a few times per day. Some of these observations can have low elevation angles and therefore signal reception can be obstructed. An example of signal availability is shown in Table I.

The proposed solution is shown in Figure 5. The proposed alternative is a combination of the two approaches. The live signals are used for receiver tests. But a copy of these signals is made in the receiver at point 2 by the debugging hardware and is sent to the PC using the same USB link as in the previous solution. The copy of the received signals is saved in a file on the PC.

Later the PC software instructs the debugging hardware tools in the receiver to substitute the digital IF stream from the front-end at point 2 with data that were recorded previously. Then the signal record is played back to the receiver. The ADC sampling clock signal is used to clock the played back IF signal samples.

An arbitrating hardware must be added to share the same USB chip by the two data streams. Also the USB chip must be steered to indicate which data stream sample is currently transfered. The USB connection (and the chip) natively can support multiple data streams in parallel. Note that the FIFO for the IF signal must support data transfer in both directions if signal playback function is required.

This setup enables repeatable tests with the same recorded GNSS signals. The recorded signals can be thoroughly analyzed for example in a PC based SDR and parameters of all signals can be determined.

This setup also allows to use much more efficiently the few signals from Galileo, COMPASS or any other of today's or future system. In the same way the signals from the GNSS

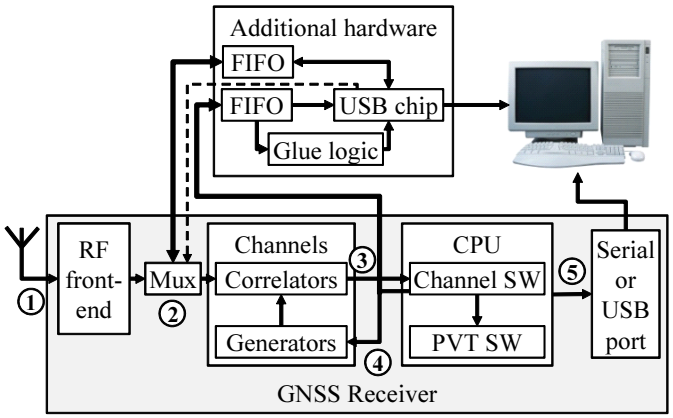

Fig. 5. The complete version of the proposed GNSS receiver debugging setup.

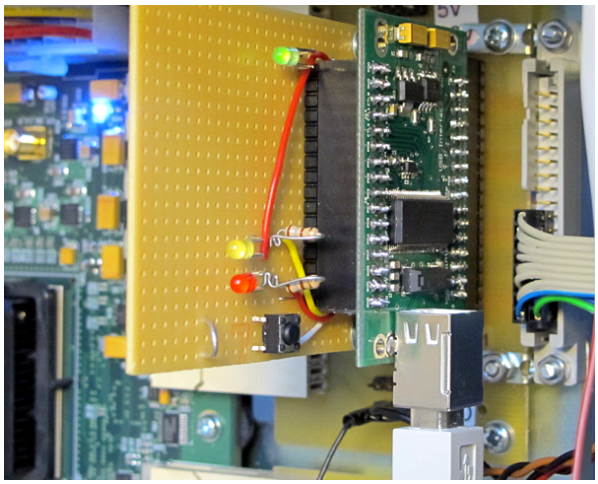

Fig. 6. The Cypress FX2 USB module attached to the FPGA board.

simulators can be reused, too. This can help in situations when access to the GNSS simulators is time limited. A set of reference test records from a GNSS simulator can be established and from that point the receiver tests can be done without access to the simulator.

It is also possible to exchange signal records between engineers or obtain them from customers to do receiver performance investigations. The contents of the received signals can be thoroughly analyzed in post-processing mode on the PC to identify the cause of the receiver problems or detailed contents of the recorded signal.

Now the possibility opens to use the pure software GNSS simulators as now the hardware link exists that can inject the simulated signals from the simulator into the signal processing chain of the receiver.

Furthermore possibilities open for experiments with receiver front-end design or optimizations, because actual front-end hardware is not required. The simulated GNSS signals or re-sampled signals from a very high quality live GNSS signal record can be used.

As the IF signal records can take a lot of storage space it is beneficial if the software running in the PC has an option to activate the recording function only when particular reception conditions or position measurements are observed. A buffer can be used in the software to store a piece of signal that was received some time before the actual event of interest was detected. This allows to investigate pre-conditions before the event of interest.

V. Implementation of the Debugging Setup

The test implementation is done in the same FPGA based platform as is used for the DGC real-time receiver. The receiver in this example is based on an ML510 FPGA board that contains Xilinx Virtex 5 FX130 FPGA chip. An L1 GNSS front-end is connected to the FPGA board. The front-end is using the Maxim 2769 GNSS front-end chip. The sampling frequency is set to the default value for this front-end which is 16.368 MHz. A serial RS-232 connection to a PC is used to receive position, measurement messages, and other status messages.

The FIFO and the glue logic are implemented in the FPGA fabric. The USB connection on the receiver side is handled by a dedicated USB data streaming chip FX2 from Cypress Semiconductor [4]. The USB chip board is attached to the FPGA board and is shown in Figure 6.

The receiver channels (an IP block) and the receiver CPU are attached to a PLB bus [3]. Internally it contains sub-buses for data and address lines and additional signals for signaling different events and commands (read, write, etc.). In the current receiver implementation the data bus is 32 bits wide and the address bus is 64 bits wide.

The proposed debugging setup captures only a necessary part of the PLB bus signals. The complete 32 data bits and 23 lowest address bits contain enough information for debugging purposes. In addition a bus flag is also captured (a single bit signal) that indicates the type of data transfer (read or write from CPU perspective). This means that in total 56 signals are captured by the debugging hardware at each data transfer from or to receiver channels.

The bus capture event is executed only when an access to the channel hardware is detected. For this purpose the chip select (CS) signal from the channel IP block is used. CPU access to other peripherals is not captured to reduce the amount of captured data.

The instance of data capture takes place in this implementation at the raising edge of the bus read or write acknowledge signal.

The Cypress FX2 USB chip supports only 16 bits wide data interface. This requires the captured 56 bit vector to be split into 16 bit wide data chunks. This is done automatically by the FIFO buffer IP block, as it supports this feature of different bus widths at input and output of the FIFO.

The chosen FIFO supports only power of 2 based ratios between data widths at input and output. For a 16 bit output constraint and the chosen captured data width the closest

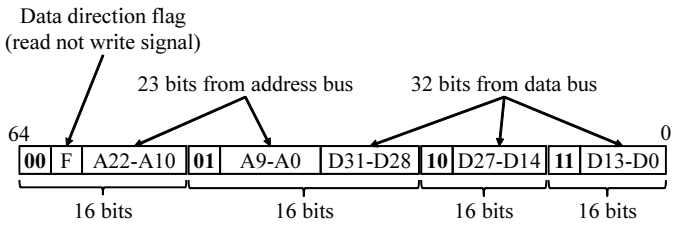

Fig. 7. The structure of the 64 bits vector at the FIFO input.

choice is a 64 bit wide FIFO input (16×4). Therefore one bus capture write to the FIFO yields four 16 bit reads at the output.

Separate clock domains were implemented to make the receiver hardware and the USB chip more independent. Presently the FIFO input is clocked by the CPU bus clock at 100 MHz. The FIFO connection to the USB chip is clocked at 30 MHz. The Xilinx FIFO IP block supports operation with two clock domains and therefore prevents data corruption when it crosses clock domains.

The four 16 bit pieces must be recombined at the PC side. There is a need for a mechanism to identify which 16 bit piece belongs to which captured bus transaction. The two most significant bits of every 16 bit transfer are reserved for this purpose.

The two bits indicate the sequence number (0 to 3) of the 16 bit transfers that belong to the same data set. The first 16 bit chunk of a bus transaction capture has this value set to 0, and the last one set to 3.

The two reserved bits reduce the amount of the useful data that can be transfered in a single capture by 8 bits. Therefore the width of the captured address bus was specifically set to 23 bits to make the total size of captured data equal to 56 bits. The final structure of the captured data is shown in Figure 7.

The current software at the PC side does processing of the incoming USB data stream in three steps which are shown in Figure 8. First it recombines the four 16 bits data chunks of each bus transaction and stores the results in two arrays. One is used to store the 32 data bus values and the other is used to store the address values. The data transfer direction bit is included in the contents of the address data.

In the second step the address information is decoded. This allows to identify to which component of the channel (and of which channel) the data in this bus transaction is associated. As a result in this step the data array contents from the first step is rearranged and stored into separate arrays for each component (code and carrier NCOs, all correlators etc.) per receiver channel (channel data structure).

The third step is the data analysis and plotting of results.

In the current setup, Matlab is used to do the three data processing steps. At the end of the processing it produces per channel figures that contain plots of correlation results, the frequency values supplied to the code and carrier generators, and the code and carrier tracking error measurements. The plot is very similar to the channel plots made by the DGC Matlab SDR [1].

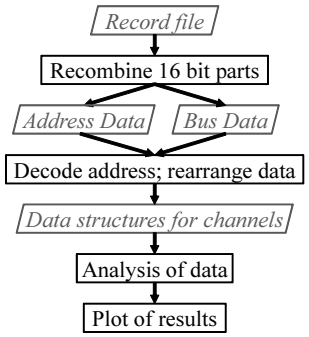

Fig. 8. The PC software data processing sequence.

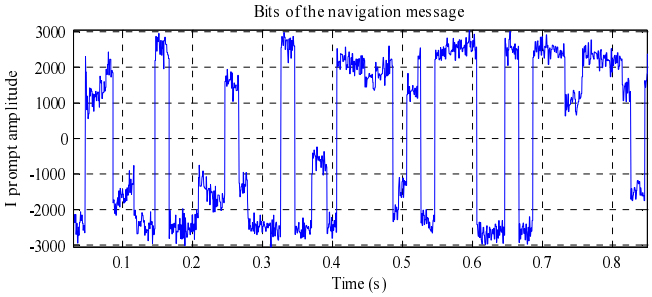

Fig. 9. The inphase prompt correlator values captured by the proposed tool at an instance of receiver performance degradation.

VI. Results

The DGC has successfully implemented the basic debugging setup. The tracking plots were created by Matlab scripts. This allowed immediately to have a much clearer picture of the nature of receiver problems. Due to size constraints of the present paper we provide only one sample plot in Figure 9. In this plot you can see the inphase prompt correlator values during a case when signal tracking performance was degraded. The picture implies that there is a strong multipath signal received by the channel, but further investigations are needed to prove that.

Due to human resource limitations DGC has not yet implemented the full debugging setup. But already the existing solution made the receiver testing much simpler as the data capture can now be done at any time instance. There was no need to write complicated checks in receiver software to activate data logging. Instead, the logging was activated from the PC whenever position plots on the PC were indicating deviations from the nominal receiver operation.

Absence of time limits for the records made it possible to capture the complete picture of signal processing and not just a small window of it. The length of the records is only limited by the size of the PC storage capacity in contrast to a few seconds when DGC was using memory buffers in the receiver to record channel activity. The current USB transfer data rate is about 40MB/min for a 14 channel receiver.

Separate tests verified the capability to capture GNSS signals from a stand alone GNSS front-end using the same USB chip, but without connection to the receiver bus capture

hardware.

It took about 3 weeks to implement the basic setup for a developer that has basic HDL development skills and a prior experience of USB data transfer design as is presented in this paper. The development time includes design and implementation of the USB chip interface hardware and Matlab code development.

The FPGA recourse use of this implementation is 2 (out of 298) RAMB36_EXPs blocks and 30 (out of 20480) slices.

Although this implementation of tools is done in the receiver FPGA, it is also possible to use a separate very small FPGA combined with the USB chip (similar to [5]). In this case the access to the receiver CPU bus and IF signals must be provided by the receiver FPGA or ASIC (see section VII-A regarding access to the signals in ASICs).

VII. Other extensions of the current solution

Current implementation of the tools is using a record file to store the captured bus data and the results are plotted after post-processing of records. In future implementations the plots (or a subset of plots) shall be done in real-time. The data rate of the captured data is not expected to cause performance issues for today's PCs.

Other applications of the presented debugging setup are possible.

For example the tools allow to monitor the increment values that are supplied to the code and carrier NCOs. The channel setup events and values are also easily detected. Furthermore it is possible to detect the channel loop closure procedure activations from the processor bus activities that are performed to service the channels interrupt signals (it was verified in tests). This yields information about receiver time.

It is possible to create (or use the same code from the receiver) software routines in the PC to obtain the raw GNSS time of transmission measurements. This allows to verify receiver measurement and position computation routines. A much more thorough testing can be done in the PC as there are no more real-time operation, receiver memory size or CPU performance constraints.

Another example is a multipath monitoring, visualizing receiver. The proposed tool is capturing data from multiple correlators per channel as the receiver CPU only reads data from these extra correlators. But the receiver CPU does not need to handle and prepare all these data according to some protocol that is used for connection through point 5. The proposed debugging hardware will do that at a high data rate without further involvement of the receiver CPU.

If multiple front-ends are used or if the sampling frequency is very high, the current USB bandwidth will be too small to transfer all these data. In this case the USB connection can be substituted with faster Ethernet or PCI Express connections.

A. Application in receiver ASICs

The complete solution is relatively easy to implement in a development receiver version, but it requires substantial number of signals to be routed to the pins of the ASIC. However some of the chips may have already most of the signals present at their pins.

If the front-end chip is implemented in a separate chip, then there is access to the digital IF samples. With some simple additional hardware it is possible to inject samples of test or recoded GNSS IF signals.

The access to the CPU bus can be complicated or impossible, but some designs of ASICs do have bus signals provided to outside world. Most often it used for connection of external memory or peripherals. However with more complex CPUs this can be of no use as such CPUs can have separate buses for memory and peripherals.

VIII. Conclusion

The authors have surveyed a set of available solutions and found that none of them completely fulfilled the GNSS receiver debugging and optimization needs. The essential information from receiver channels was severely time limited or did not show the full picture.

The proposed solutions provide great improvement in visualization of receiver signal processing activities and already has proved to be very helpful in receiver debugging.

After the first tests it was noted that the combination of an FPGA based receiver and the proposed tools combines the good qualities of hardware and PC based software defined receivers. The GNSS signal processing is done at the speed of a hardware receiver, but the signal processing visualization is done with near SDR flexibility. For example it can take about 2 hours to process 1 minute of a GNSS signal in a pure Matlab GPS SDR. By the proposed tools it takes one minute to receive the signal and a couple of minutes to process the captured data.

These are very early conclusions, as the first tests were finished a shortly before publication of this paper.

Acknowledgment

The authors acknowledge the financial support granted by the Danish National Advanced Technology Foundation under J.no. 009-2007-2.

References

[1] K. Borre, D. M. Akos, N. Bertelsen, P. Rinder, S. H. Jensen , *A Software-defined GPS And Galileo Receiver. A single frequency approach*. Boston, MA: Birkhäser Boston, 2006.

[2] E. D. Kaplan and C. J.Hegarty, *Understanding GPS. Principles and applications*, 2nd ed. Boston, MA: Artech House Publishers, 2005.

[3] *CoreConnect Architecture—Processor Local Bus (PLB)*, Xilinx, Inc. Available: http://www.xilinx.com/ipcenter/processor_central/coreconnect/coreconnect_plb.htm

[4] *EZ-USB FX2LP Overview*, Cypress Semiconductor. Available: http://www.cypress.com/?id=193

[5] *E-Series FPGA Cards*, Pico Computing. Available: http://www.picocomputing.com/e_series.html